ICASE

SAGA

A System To Automate the Management of Software Production

(NASA-CR-185784) SAGA: A SYSTEM TO AUTOMATE THE MANAGEMENT OF SOFTWARE PRODUCTION (ICASE) 14 p

N90-70096

Unclas

00/61 0224387

Roy H. Campbell

and

Paul G. Richards

Report No. 80-36

December 16, 1980

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

SAGA

A System to Automate the Management of Software Production

Roy H. Campbell
University of Illinois

and

Paul G. Richards
University of Illinois

ABSTRACT

SAGA is a software development system designed to integrate software production tools and techniques into a flexible management system through the use of special attributed grammars to represent management schemes. Both the software life cycle and project components are described by formal grammars. This formalization will aid understanding of management techniques for complex projects and encourage the automation of repetitive and tedius managerial tasks. Project direction and management is monitored by SAGA from inception through completion and allows identification and scheduling of critical events along with integration of project specification, design, implementation, certification, and maintenance. The SAGA project is expected to have a particularly positive impact on quality software production for reliable computer applications.

This research was partially supported by NASA Grant NSG-1471 at the University of Illinois and by NAS1-14472 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

NASA Contract

1 Introduction.

The process of software design, production, and maintenance follows a pattern of activity often referred to as the software lifecycle. The management of the lifecycle is critical to the success of the eventual software product [Brooks, 75]. The response to this critical management problem has been the evolution of Software Engineering. Software Engineering has improved the production of high-quality software through the use of new tools [Jensen & Tonies, 79]. Methodologies to improve design, programming, and maintenance have also been proposed. The effectiveness of these tools depends on the proper management of information they produce. The SAGA project proposes a method of integrating many of these software production tools and techniques into a flexible, formal management system for software development.

1.1 Goals.

The goal of the SAGA project is the development of a formal management methodology and system that will enhance production of complex, reliable, certifiable software. Complex software is typified by long development, interaction between several developers, complex module interfaces, and requirements for auditing, certification, and quality. We view software management as the recognition of valid sequences of (perhaps concurrent) activities in a software lifecycle. Certification of a software product corresponds to the recognition of a valid history of software development, verification, and validation. A satisfactory system for development of such software requires:

- Recognition of valid sequences of activities in software lifecycle.

- Consistent application of a well defined management policy throughout the life of the project.
- The ability to ascertain development status of the software project, and retrieval of project information.
- Automation of repetitive and tedious management tasks (e.g., auditing, version control).
- Automated identification of intermodule dependencies between and within project phases.

Desirable attributes of this kind of software development system include:

- Machine processable specification, design, and implementation languages.
- Centralized and coordinated storage and processing of all project information (e.g., requirements, specifications, designs, data, source and object code, testing information, documentation, design decisions).
- Appropriate communication and documentation tools (e.g., interuser communication, intraproject communication, communication between users and the management system).
- Ability to integrate already developed tools (e.g., automated program verifiers, test data generators, optimizers, performance analyzers).
- Checking of the consistency of intermodule dependencies.

1.2 Other Systems.

Many systems have been developed to aid software development. Among them are CADES [Pratten, 78], Bell Labs' Unix/Programmer's Work Bench [Dolotta, et al., 78], and Gandalf [Habermann, 79]. Each of these systems satisfies some of the criteria above.

1.2.1 CADES.

The CADES (Computer Aided Development and Evaluation) system was developed at International Computers Limited as an operating system development aid. It is composed of a database, a language interface called SDL, and a formalism for transformations of problems called "Structural Modeling". Structural modeling specifies the relations between data and the objects that manipulate the data and supports refinement of both the relationships and the objects until an implementation is realized. The database is used to store these relationships and maintain auditing history on the refinements that have been applied. CADES provides facilities for project organization, version control, interfaces to compilers/linkers for automatic invocation after module modification, and facilities for including other tools for further analysis of the project database.

1.2.2 PWB.

The Programmers Work Bench is an adaptation of the UNIX operating system to the needs of large software development projects. PWB provides an efficient programming environment that is separated from the system on which the programs are to be executed. It provides additional tools for software development, including a Source Code Control System [Rochkind, 75] and remote job entry software. Unix provides facilities for editing & file storage, document preparation, and some user communications. SCCS provides version control and auditing of modules of source code, documentation, or test data.

1.2.3 Gandalf.

Gandalf is an interactive software development system for the ADA programming language. A syntax oriented editor permits entry of programs. The language INTERCOL and software development control facility of Gandalf is described in [Tichy, 80]. INTERCOL is one of a class of languages known as Module Interconnection Languages. INTERCOL represents the structure of systems by describing module interfaces. Interface consistency is maintained by type checking between modules and notifying appropriate developers when inconsistencies are found. INTERCOL also has the ability to describe multiple versions of software using a concept of "families" of modules and systems.

1.3 Analysis of Other Systems.

No system presently solves the problem of managing software development in a complete and satisfactory manner. The isolated collection of tools in PWB require the programmer to remember important procedures and to use the tools correctly. Structure-based systems such as INTERCOL and CADES attempt to restrict the software development process to ensure that inter-module interfaces are correct and consistent. Neither system integrates the restrictions with the target source code or extends automated management to all phases of the project. Our proposal combines the various software development system approaches into a flexible and effective system.

2 The SAGA System.

2.1 Approach.

The SAGA system provides an integrated approach to the management of the software production process by combining various existing techniques for recognizing, representing, and analyzing formal specifications. Its primary components are:

- Formal representation of management policy by management grammars (LALR(1) 'attributed' BNF grammars which use events in the software life cycle as terminal symbols).
- Primitives for specifying module structures, system structures, and events that occur in the lifecycle.
- A central database with provisions for storage of all project related information.
- An inter-project library for sharing code, data, and development procedures.
- Formal specification and constraint of database manipulation(s) via development grammars. The formal specification allows such features as automatic recompilation and auditing.
- Formal representations of and uniform interfaces between specification, design, and implementation languages to permit mechanical consistency checking within and between phases in the software life cycle.
- User oriented communication facilities that include not only the ability to "talk" and "mail" between users but also archival notesfile facilities to record policy decisions and allow discussions to take place in writing in a machine readable form.

The management and development grammars specify the sequence of acceptable events in the software project from its inception through its completion. Events can be generated by programmer interaction or by the partial parsing of a sentence of a grammar. The management grammar represents policy and its terminal symbols are events generated by the programmer or by the system. As sentences of the language specified by the management grammar are parsed, different management primitives are invoked. These primitives can start subtasks controlled by other management grammars, declare events to higher level tasks, or invoke specific software tools controlled by development grammars. The development grammars are used to control access to specific tools such as compilers and editors. This hierarchical management system can be used to configure complex and concurrent project development schemes. Tracing of the parsing can be done to any granularity, thus allowing auditing to any level, and the construction of complete, detailed records of activity completion.

The database manipulation routines have, as integral components, bookkeeping routines which audit intermodule references and ensure consistency through the project. This level of bookkeeping is required to simplify recognition of unconsidered specifications or uncoded designs and issue requests for their completion.

Information in the SAGA system should be represented in a machine processable form, i.e., high level language. This allows identification of the events that are considered important to the management policy. It is also expected that investigations into automated analysis of project phases such as validating a design for consistency with its specification will require that specifications and design be represented in a high level language. We believe the SAGA database could be a useful tool for the future development of such automated analysis. Management and development grammars based on the syntax of the specification, design, and implementation of high-level languages allow SAGA to control project development to the individual statement level if required.

2.2 Applications.

The management schemes employed in a SAGA development system are intended to enhance human engineering aspects of software production. For software development of applications which must be very reliable, the management schemes can impose a precise and rigid development discipline. Alternatively, SAGA might provide an environment for rapid development of scientific or research programs that do not require such rigid control. The system can be used to enhance the productivity of the system developer by providing on-line project information, coordinating efforts and module sharing among teams, prompting for completion of standardized documents, cross referencing between phases of the project, providing status reports of the project, and sharing modules between projects.

Since the management policies for the project are explicitly stated (by the management grammars and the attributes on the project languages) and the policies are enforced by the software tools themselves, validation of software produced under the system is much simpler. It is possible to log every operation taken during the development in order to satisfy strict auditing procedures. Formal structuring of the development process may allow more rigid validation assumptions to be made (by either automatic or manual theorem provers) about specifications and coding of modules.

The SAGA system is designed to direct programmer activity without imposing excessive restraint. It is expected that managers will recognize the need for balance between programmer control and freedom. SAGA provides an excellent vehicle for experimentation in various management policies and can be used to analyze the effects of those policies.

2.3 Example.

Below are some grammar fragments for a hypothetical SAGA system that control updates to project source code. A single management grammar specifies version and release policy. A development grammar controls updates to source modules. The grammar is represented using the usual BNF meta-symbols {} to denote repetition. [x] indicates semantic action "x" is to be performed (described in the narrative below). Terminal symbols which are events are represented as upper case letters between quotes:

<u>The Management Grammar</u>

```
<new version> ::= <initialize modify> <modify module> <release>

<initialize modify> ::= "NEW_VERSION" [1]

<modify module> ::= { <new code> <validation & verification> }+

<new code> ::= { "CODE" [2] }+ "CODE_COMPLETE" [3]

<validation & verification> ::= "VERIFY" [4] "VERIFY_COMPLETE"

<release> ::= "RELEASE" [5]
```

<u>The Development Grammar</u>

```
<code> ::= { { <module work> }+ <check consistency> }* "DONE" [6]

<module work> ::= "EDIT" [7]

<check consistency> ::= "CHECK" [8]
```

The events used by this example are described below:

NEW_VERSION - The project manager wishes to authorize a change to a source module, and declares this event.

CODE - The project manager indicates changes may proceed by declaring this event.

CODE_COMPLETE - This event is requested by the project manager when all requests for new coding have been made.

VERIFY - Project manager indicates that the modification is approved and testing should start by declaring this event.

RELEASE - Project manager can release the tested module as a new version by declaring this event.

DONE - A programmer declares this event when all modifications to the module are complete.

EDIT - Raised by invoking an editor on the source module.

CHECK - Programmer uses a compiler or analyzer on the source module.

Under a SAGA system using these grammars, a possible sequence of events is described below:

i) Someone requests a change to a source module. The Project manager indicates that a change is to occur by declaring the "NEW_VERSION" event. Management primitives invoked at [1] request a reason and description of the change from the project manager, which will be stored in the new version's documentation.

ii) The project manager declares one or more "CODE" events. Primitives invoked at [2] request the name of the module and the programmer assigned to make the change. A temporary copy of the module is created. The programmer is authorized to use the development grammar to access the temporary copy. The system uses the development grammar independently and asynchronously of the management grammar. After the "CODE_COMPLETE" event, the management grammar primitive at [3] waits for all coding subtasks to complete.

iii) The programmer invokes the editor, which declares "EDIT". Primitives at [7] check his authorization and allows the editor to proceed.

iv) After editing, the programmer uses a compiler to check the source module for errors. This invokes primitives at [8] that make sure no undefined subroutines are referenced.

v) After the programmer is satisfied that the changes are correct, he declares the "DONE" event, which causes primitives at [6] to terminate the subtask using the development grammar.

vi) The project manager is notified that the modules are changed, and is allowed to declare event "VERIFY" to start verification of the module. Primitives at [4] start another development grammar for verification. The project manager waits until the "VERIFY_COMPLETE" event is declared by the verification subtask.

vii) Verification is completed and "VERIFY_COMPLETE" is declared. The manager is notified that the module is ready, and declares the "RELEASE" event.

Primitives at [5] make the temporary copy into a new release in the database catalog, notifying the appropriate users that the new release is complete.

3 Prototype.

A prototype SAGA system is being implemented in Pascal. It will permit the entry of project information and control of the development process in both a batch and timesharing environment. An interactive editor will promote immediate capture of programming and design decisions. A subset of the commands available to the interactive editor will be designed to process batch or file oriented input to permit remote preparation of large volumes of program material or use of existing material. Editors using table driven LALR(1) parsers will allow entry and recognition of requirements, designs, management, and programs. Editor commands will allow manipulation of the parse trees [Teitelbaum, 79]. The parsers will drive a syntax directed translation scheme associated with each grammar. The translation scheme will interface to management primitives that control the database and the information it contains. Initially, the management primitives will be encoded as Pascal code segments.

SAGA editors will be constructed automatically using two tools: a table generator and skeleton editor. This approach will facilitate experimentation with new specification, design, and management languages while ensuring reliable implementation. New programming languages can be included in a SAGA system with minimal overhead. The prototype SAGA system includes both a generator and a skeleton editor.

The prototype SAGA software management system for PASCAL is being constructed to test the effectiveness of the system and will include management grammars and the use of example requirements and design languages. In particular, the management grammars will support control of versions and concurrent project development activities. Various tools such as the project data base, compilers, mail, notes files, and documentation preparation systems will be integrated into the eventual prototype SAGA system for PASCAL.

4 Conclusion.

The formalization of management in software development projects will improve understanding of the project lifecycle and strengthen the validity of software certification. The SAGA system provides an approach to the automatic generation of software development systems and the eventual formalization of management schemes. Management of the development process can be applied to all interactions and information in the project from the moment of entry to the computer through its lifecycle.

Although considerable research and development is required to realize a production version of SAGA, the prototype specification suggests that such systems can be constructed and that management schemes for the production of software can be described using augmented grammars. We welcome comments, suggestions, and examples of management schemes that have been applied to actual software production projects..

ACKNOWLEDGMENT

The authors would like to acknowledge the helpful comments of Martin McKendry and John Knight.

6 References.

[Brooks, 75] Brooks, F.P., The Mythical Man Month, Addison Wesley, Reading, MA., 1975.

[Dolotta, et al, 78] Dolotta, T.A., R.C. Haight, and J.R. Mashey, "The Programmer's Workbench", Bell System Technical Journal, Vol. 56, No. 6, July-August 1978, pp. 2177-2200.

[Habermann, 79] "The Gandalf Project", Presentation at the Software Tools Workshop, Pingree Park, Colorado, May 1979.

[Jensen & Tonies, 79] Jensen, Randall and Charles Tonies, Software Engineering, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[Pratten, 78] Pratten, G.D., "The CADES Software Development System", Internal Document, International Computers Ltd, Kidsgrove, Stroke-on-Trent, England, 1978.

[Rochkind, 75] Rochkind, M.J., "The Source Code Control System", IEEE Transactions on Software Engineering, SE-1, December 1975, pp. 364-370.

[Teitelbaum, 79] Teitelbaum, T., "The Cornell Program Synthesizer: A Syntax Directed Programming Environment", SIGPLAN Notices, Vol 14, No. 10, October 1979

[Tichy, 80] Tichy, Walter F., "Software Development Control Based on System Structure Description", Carnegie-Mellon University Department of Computer Science Technical Report CMU-CS-80-120, January 1980.